

## Lec: 7

### Advanced Optimization Algorithms

- These algorithms are able to get logistic regression to run much more quickly than it's possible with gradient descent (No Need to manually pick  $\alpha$ ) (Advantage).
- They also scale much better to very large machine learning problems, such as IF we had a very large number of Features. (Advantage)
- More Complex (disadvantage)

→ In Optimization Algorithms we use ready code (ready Functions) in Matlab called "fminunc". This code can be used with Linear regression, Logistic regression .... this means that optimization algorithms don't care about the Algorithm type as it only need the code that can compute  $J(\theta)$  and  $\frac{\partial}{\partial \theta_i} J(\theta)$

### Optimization Algorithms:

- Gradient Descent
- Conjugate Gradient
- BFGS
- L-BFGS

Fminunc attempts to Find a minimum of a scalar Function of several Variables Starting at an initial estimate.

```
return      return      return
[Opt Theta, FunctionVal, exitFlag] = fminunc(@ CostFunction, InitialTheta, options);
= fminunc (@ CostFunction, InitialTheta, options);
```

The Function arguments:

1. The Objective Function to be minimized (the Cost Function)
2. The initial values of the Function Variables (parameters  $\theta$ )
3. The Optimization Options specified in Options.

The Function return:

1. The Optimal values of the Function Variables (parameters  $\theta$ ) at which the local minimum of the Function.
2. The value of the objective Function at the optimal values of the parameters.
3. A value exitFlag that describes the exit condition.

Example:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}, J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

We need to get  $\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$  and  $\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

Code

```
Options = optimset('GradObj', 'on', 'MaxIter', 100);
```

```
InitialTheta = zeros(2, 1);
```

```
[OptTheta, FunctionVal, exitFlag] =
```

```
fminunc(@CostFunction, InitialTheta, Options);
```

```
Function [jVal, gradient] = CostFunction(theta)
```

```
jVal = (theta(1) - 5)^2 + ... (theta(2) - 5)^2;
```

```
gradient = zeros(2, 1);
```

```
gradient(1) = 2 * (theta(1) - 5);
```

```
gradient(2) = 2 * (theta(2) - 5);
```



# Code In General

→  $\text{Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$  : 1<sup>st</sup> Step Calculate Cost Function

Function [jVal, gradient] = CostFunction

$$jVal = [ \text{code to compute } J(\theta) ]_j$$

Gradient (1) = [ Code to compute  $\frac{\partial}{\partial \theta_0} J(\theta) ]$ ;

gradient (2) = [ Code to Compute  $\frac{\partial}{\partial \theta_1} J(\theta) ]$ ;

...

gradient(n+1) = [code] to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$

2<sup>nd</sup> Step determine the Optimization Options

Options = optimset('GradObj','on','MaxIter','100');

This Function determine the optimization options, the most important options are the state of the optimization objective (we choose gradient descent as an optimization objective) we make the state 'on' to make it able to be used. AND the maximum number of iterations

For Simplification, we always make Gradobj 'on' and MaxIter '100'.

### 3<sup>rd</sup> Step Initial Values of $\phi$ "zeros"

Initial Theta = Zeros (  $\overset{\text{number of row}}{\text{number of } \odot \text{ (n)}}, \underset{\substack{\downarrow \\ \text{number of column}}}{1} );$

## 4<sup>th</sup> Step get Fminunc Function

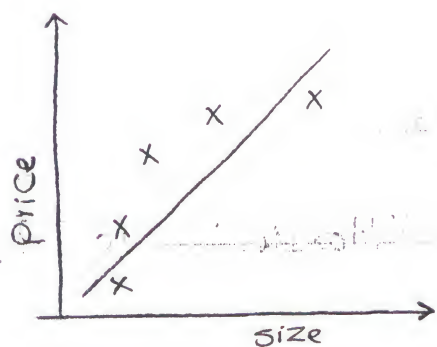
[OptTheta, FunctionVal, exitFlag] --

= fminunc (@CostFunction, InitialTheta, Options);

→ This Function takes 1. CostFunction 2. InitialTheta 3. Options and gives Optimal value of theta that gives the optimal solution, Optimal value for CostFunction (equals to zero at perfect solution) AND returns exitFlag which value equals to 1 when the code finishes running.

## Regularization

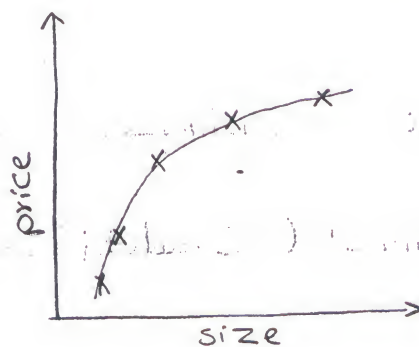
Example: Linear regression (housing Prices)



$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Under Fitting

here, we don't fit the data at all.

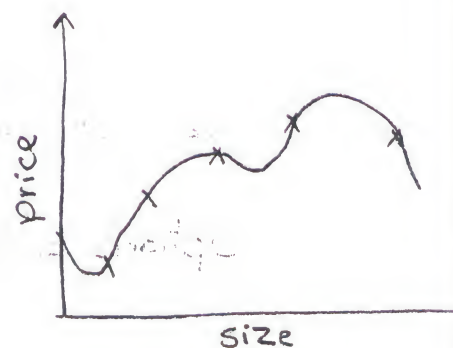


$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Just Right

here, we just fit this data problem we may not be able to get the right answer

For another data,  
(The Best Way For Fitting Data)



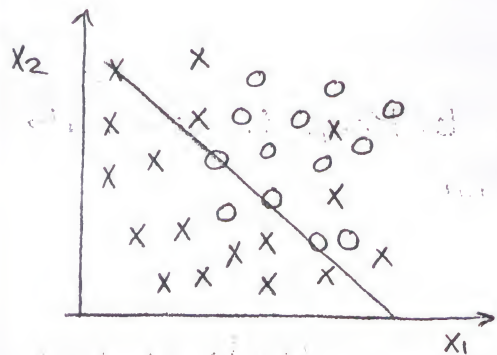
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Over Fitting

here, we get the best fitting for this data but we can't fit any other data!!



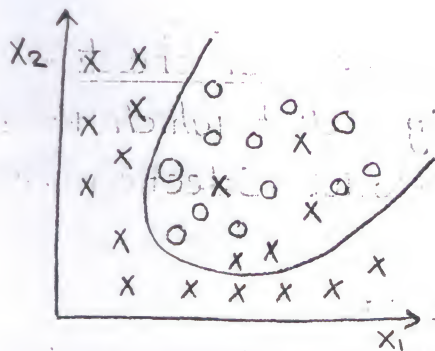
## Example : Logistic regression



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

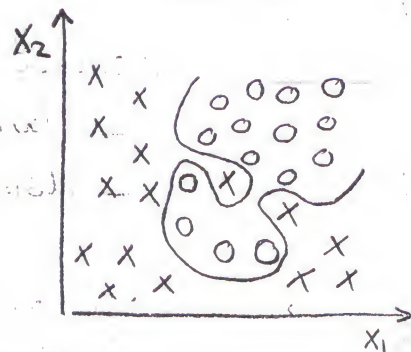
where  $g$  : Sigmoid Function.

Under Fitting



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$

Just Right



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \dots)$$

Over Fitting

Overfitting : IF we have too many Features, the learned hypothesis may fit the training set very well

( $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \approx 0$ ), but Fail to generalize to new examples (predict prices on new examples).

→ Then, Overfitting occurs when we have too many Features

For the First example (housing prices) let

$x_1$  = size of house

$x_2$  = no. of bedrooms

$x_3$  = no. of floors

$x_4$  = age of house

$x_5$  = Kitchen size

⋮

$x_{100}$

## Two Ways To Solve This Problem.

### 1. Reduce Number Of Features.

- Manually select which features to keep (most important)
- Using Model Selection Algorithm.

### 2. Regularization

- Keep all the features, but reduce magnitude (values) of parameters,  $\theta_j$ .

For housing Price Example.

→ Just Right  $\Rightarrow h_0(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

→ Overfitting  $\Rightarrow h_0(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

to solve the problem of overfitting using regularization we

reduce values of  $\theta_3$  and  $\theta_4$  so  $\theta_3 x^3 + \theta_4 x^4 \approx 0$ .

So we return to "Just Right" (Best Fitting way).

Note: we reduce these parameters  $\theta_j$  during iteration. This means that we use them at the early number of iteration and start to reduce them step by step.

- Works well when we have a lot of features, each of which contributes a bit to predicting  $y$ .

→ we will study Regularization in details.



## → Cost Function.

### • Regularization.

Small values for parameters  $\theta_0, \theta_1, \dots, \theta_n$

- "Simpler" hypothesis

- less prone to overfitting

Features:  $x_1, x_2, \dots, x_n$

Parameters:  $\theta_0, \theta_1, \dots, \theta_n$

Cost Function  $J(\theta)$

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

+ Goal:  $\min J(\theta)$

\* The extra regularization term at the end is used to shrink every single parameter  $\theta$

$$\left[ \lambda \sum_{j=1}^n \theta_j^2 \right]$$

•  $\lambda$  is the regularization parameter that controls a trade off between 2 different goals:

→ The First goal objective, is that we would like to fit the training data well.

→ The Second goal is, we want to keep the parameters small and therefore keeping the hypothesis relatively simple to avoid overfitting.

The value of  $\lambda$  mustn't be very large or very small  
the best value of  $\lambda$  is from 0 to 0.99

## → Regularized linear regression.

• Cost Function

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

• Gradient descent

$$\theta_j := \theta_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

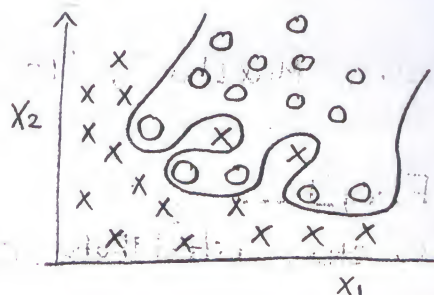
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

(j = 1, 2, 3, ..., n)

}



## → Regularized Logistic regression



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^2 + \dots)$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

### 1. Cost Function

$$J(\theta) = \left[ -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

### 2. Gradient descent

Repeat for  $i = 1, 2, \dots, n$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

$$(j = 1, 2, 3, \dots, n)$$

3

## • Advanced optimization.

### Code

options = optimset('GradObj', 'on', 'MaxIter', '100');

InitialTheta = Zeros(n, 1);

[optTheta, FunctionVal, exitFlag] ...

= fminunc(@CostFunction, InitialTheta, options)

Function [jVal, gradient] = CostFunction(Theta)

jVal = [Code to compute  $J(\theta)$ ];

$$J(\theta) = \left[ -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

gradient(1) = [Code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ];

$$\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

gradient(2) = [Code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ];

$$\frac{\partial}{\partial \theta_1} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)} - \frac{\lambda}{m} \theta_1$$

⋮

gradient(n+1) = [Code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$ ];